FLEXIBLE ACCESS CONTROL FOR CAMPUS AND ENTERPRISE NETWORKS

A Thesis
Presented to
The Academic Faculty

by

Ankur Kumar Nayak

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in
Computer Science

School of Computer Science
Georgia Institute of Technology
May 2010

FLEXIBLE ACCESS CONTROL FOR CAMPUS AND ENTERPRISE NETWORKS

Approved by:

Professor Nick Feamster, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Russell Clark
School of Computer Science
Georgia Institute of Technology

Professor Ellen Zegura
School of Computer Science
Georgia Institute of Technology

Date Approved: 5 April 2010

To my family, for their support and encouragement.

ACKNOWLEDGEMENTS

This thesis has been made possible by the constant presence and invaluable guidance of numerous people. I would, therefore, like to show my gratitude towards them.

I feel extremely pleased in thanking **Prof. Nick Feamster**, without whose support and able guidance, this thesis would have never been completed. His tremendous dedication to work, great intellect, effective time management, backed by an equal magnitude of hardwork has left an ever lasting impression on me. Working with an immensely skilled person like him was not only full of enjoyment and learning but a great experience as well. He will always be a source of inspiration for me.

I am also grateful to **Dr. Russell Clark**, for his constant co-operation and expert inputs throughout the course of our work.

This work wouldn't have been possible if not for the efforts of my fellow researcher and good friend **Hyojoon Kim**. I'd always remember some of the sleepless nights we spent working on the project, debugging issues, fixing unimaginable bugs and what not. I wish him me the very best for his future. I would also like to thank my friends Bilal, Pushkar, Murtaza, Yogesh, Amit, Vishal and Sunil for all the fun times we spent. My final vote of thanks goes to all the members of the NTG lab who helped me in every possible way to make my work easy.

I dedicate this thesis to my parents, for everything they have showered upon me, to my family, for their constant support and encouragement, and to the Almighty.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

LIST OF TABLES vii

LIST OF FIGURES viii

SUMMARY ix

I INTRODUCTION 1

1.1 Motivation 1

1.2 Resonance: Flexible Access Control for Campus and Enterprise Networks 2

1.3 Contributions 4

II BACKGROUND AND RELATED WORK 5

2.1 Background 5

2.1.1 Access Control and Monitoring 5

2.1.2 Overview of Existing System 5

2.1.3 Problems with the current design 7

2.2 OpenFlow 8

2.3 NOX: A controller framework 10

2.3.1 Programmatic Interface 11

2.4 Related Work 12

III RESONANCE: DESIGN AND USE CASES 15

3.1 Overview 15

3.2 Formal State Machine Model for Access Control 16

3.3 State machine based policy specification 18

3.4 Network Admission Control: Redesigning START 20

3.4.1 State Description 21

3.4.2 State Transitions 22

3.4.3 Resonance Step-by-Step 23
3.5 Dynamic Rate Limiting 26
IV IMPLEMENTATION, DEPLOYMENT AND PRELIMINARY EVALUATION 29
4.1 Implementation 29
4.2 Deployment 32
4.2.1 Connectivity and Setup 33
4.3 Preliminary Evaluation 36
4.3.1 Flow table size analysis 36
4.3.2 Flow setup and performance experiments 38
4.4 Challenges 42
V CONCLUSION 44
APPENDIX A RESONANCE SCREENSHOTS 46
REFERENCES 50

LIST OF TABLES

1 The header fields that an OpenFlow switch can map. 9

2 Flow table entries for Registration and Operation states. 22

3 Flow table entries for Scan and Quarantined states. 23

LIST OF FIGURES

1 Current START Architecture. 6

2 OpenFlow-enabled Switch. The Flow Table is programmed by a controller over the Secure Channel. 10

3 Components of a NOX-based network: OpenFlow (OF) switches and a server PC running NOX with different components. Communication among components and NOX happens using events and message passing. 11

4 State transitions for a host. The controller tracks the state of each host and updates the current state according to inputs from external sources (*e.g.*, network monitors). 24

5 Applying Resonance to START. 25

6 Flow chart representing sequence of actions in Resonance when applied to START. 26

7 State transitions for dynamic rate limiting. 27

8 Event handling and message passing between Resonance and Messenger components. 31

9 Current research testbed with connectivities between three campus buildings. 34

10 Number of concurrent flows in subnet A. 37

11 Number of concurrent flows in subnet B. 38

12 Flow setup time delay using Resonance. 39

13 Ping delay in Resonance. 40

14 Secure copy delay in Resonance. 41

15 GEC7 Demo setup. 46

16 Resonance login page. 47

17 Successful login. 48

18 User redirected to original website. 49

SUMMARY

We consider the problem of designing enterprise network security systems which are easy to manage, robust and flexible. This problem is challenging. Today, most approaches rely on host security, middleboxes, and complex interactions between many protocols. To solve this problem, we explore how new programmable networking paradigms can facilitate fine-grained network control. We present *Resonance*, a system for securing enterprise networks , where the network elements themselves enforce dynamic access control policies through state changes based on both flow-level information and real-time alerts. Resonance uses programmable switches to manipulate traffic at lower layers; these switches take actions (*e.g.*, dropping or redirecting traffic) to enforce high-level security policies based on input from both higher-level security boxes and distributed monitoring and inference systems. Using our approach, administrators can create security applications by first identifying a state machine to represent different policy changes and then, translating these states into actual network policies. Earlier approaches in this direction (*e.g.*, Ethane, Sane) have remained low-level requiring policies to be written in languages which are too detailed and are difficult for regular users and administrators to comprehend. As a result, significant effort is needed to package policies, events and network devices into a high-level application. Resonance abstracts out all the details through its state-machine based policy specification framework and presents security functions which are close to the end system and hence, more tractable.

To demonstrate how well Resonance can be applied to existing systems, we consider two use cases. First relates to "Network Admission Control" problem. Georgia

Tech dormitories currently use a system called START (Scanning Technology for Automated Registration, Repair, and Response Tasks) to authenticate and secure new hosts entering the network [23]. START uses a VLAN-based approach to isolate new hosts from authenticated hosts, along with a series of network device interactions. VLANs are notoriously difficult to use, requiring much hand-holding and manual configuration. Our interactions with the dorm network administrators have revealed that this existing system is not only difficult to manage and scale but also inflexible, allowing only coarse-grained access control. We implemented START by expressing its functions in the Resonance framework. The current system is deployed across three buildings in Georgia Tech with both wired as well as wireless connectivities. We present an evaluation of our system's scalability and performance. We consider dynamic rate limiting as the second use case for Resonance. We show how a network policy that relies on rate limiting and traffic shaping can easily be implemented using only a few state transitions. We plan to expand our deployment to more users and buildings and support more complex policies as an extension to our ongoing work.

Main contributions of this thesis include design and implementation of a flexible access control model, evaluation studies of our system's scalability and performance, and a campus-wide testbed setup with a working version of Resonance running. Our preliminary evaluations suggest that Resonance is scalable and can be potentially deployed in production networks. Our work can provide a good platform for more advanced and powerful security techniques for enterprise networks.

CHAPTER I

INTRODUCTION

We present a dynamic and flexible access control system which is capable of supporting diverse and complex network policies. With increasingly many diverse network hosts and devices, network management and security has become challenging and error-prone [15]. All the state-of-the-art techniques require manual intervention and are inflexible and coarse-grained. To solve this problem, we incorporate and evaluate a centralized state-machine based model, Resonance [18], which leverages programmability in today's switches to tackle these problems. Resonance is currently functional on a research testbed across a subset of buildings in the Georgia Tech campus and we plan to replace the current access control technology with our system.

1.1 Motivation

Enterprise networks comprise of various network elements and host many heterogeneous and potentially untrusted devices. Even with significant advances in host security, the growing number and types of these devices pose a significant burden on network administrators to be continually aware of these devices and to be able to detect and isolate potential security threats. These devices run a variety of operating systems and have different softwares bringing with them a diverse set of vulnerabilities. Large-scale network attacks such as botnets or DoS present another layer of security breach on top of host-level threats. In fact, network-level compromises can be more lethal than most host-level threats as these spread much faster and can potentially bring down an entire network if unchecked. In the face of these challenges, it becomes imperative to be able to monitor the entire network continually and identify and isolate various network threats as early as possible.

Today, network administrators resort to many different ad-hoc techniques to safeguard hosts from attacks. A variety of devices are deployed to cater to specific security aspects of the network. To name a few, there are firewalls, security middle-boxes, VLANs, captive portals, Radius servers, VMPS, etc. There are several different ways of just implementing a simple captive portal authentication system, ranging from DNS-based redirection to VLAN-based host isolation. These techniques are too static, unscalable, and rigid; moreover, none of the existing techniques address the issue of continuous monitoring of the entire network and isolation of these machines on the fly. As an instance, DNS-based redirection techniques often involve setting timers to re-enable normal DNS after successful authentication. Any change in timer values or late authentication response can cause the system to behave erroneously. In addition, most of the threat identifications and removal is done manually, often resulting in human errors.

1.2 Resonance: Flexible Access Control for Campus and Enterprise Networks

Instead of placing trust in the end hosts or relying on security middleboxes, an enterprise network should offer mechanisms that directly control network traffic according to dynamic, fine-grained security policies, and in response to input from distributed network monitors. Extending the metaphor of a network operating system [13] to the design of *secure* networks, we present the design of *Resonance*, which provides mechanisms for directly implementing dynamic network security policies in the network, at devices and switches, leaving little responsibility to either the hosts or higher layers of the network. We draw inspiration from the design of secure operating systems, where complex system components are built using small, hardened, trusted components as a base. Similarly, Resonance imbues the network layer with the basic functions needed to implement security policies, as well as a control interface that allows monitoring systems to control traffic according to predefined policies.

As in previous work (*e.g.*, Ethane [3]), Resonance controls traffic using policies that a controller installs in programmable switches [20, 5]. Extending this paradigm, we create a *dynamic access control* framework that integrates the controller with monitoring subsystems. This integration allows an operator to specify how the network should control traffic on an enterprise as network conditions change. For example, Resonance can automatically quarantine hosts or subsets of traffic when a compromise or other security breach is detected.

Recent trends enable integration of dynamic monitoring and control. First, *programmable (and software-based) network devices* [20, 5] allow more direct, fine-grained control over network traffic. At first blush, programmable network devices might seem to present yet another source of complexity, but we believe that this programmability actually presents an opportunity to proactively secure the network layer. Second, *distributed network monitoring algorithms* can now quickly and accurately correlate traffic from many distinct (and often distributed) sources to detect coordinated attacks (*e.g.*, for detecting botnets [12] and spammers [21]). Finally, the trend towards logically centralized network control [11, 8] allows us to more easily integrate distributed network monitoring with dynamic network control.

Resonance's design follows a state-machine model. A network system can be represented using a state machine abstraction. A state machine is defined in terms of a set of states and their transitions based on inputs from the network. In the context of network access control, each state represents a set of policies stating which network services are allowed in that state and which are not. The definition of a state can be made more general by extending it to not only include network services but also rate limiting and other network parameters. For example, hosts belonging to certain states are only allowed to send traffic at a particular rate. This flexibility makes this state machine model a powerful tool to control the network. Several enhancements and state machine variations can be incorporated into the model.

1.3 Contributions

In this thesis, we demonstrate a flexible and dynamic network access control model, present our implementation and deployment experiences at Georgia Tech campus testbed, show our initial evaluation results, and analyze two different use cases where our system can be applied. As far as we know, Resonance is the first system that allows dynamic, fine-grained network policies based on integration with monitoring systems. We explore how Resonance not only simplifies the implementation of network security policies but also enables fine-grained security policies and a wide range of features. Our evaluation studies suggest that Resonance can scale well with a good performance. Our campus-wide testbed itself presents a great platform for setting up and evaluating new protocols and research ideas concurrently. We believe that our current efforts can serve as a great starting point for the design of next-generation access control systems.

CHAPTER II

BACKGROUND AND RELATED WORK

2.1 Background

We describe the network access control problem in the context of the Georgia Tech campus network. We also introduce OpenFlow [20], an interface for programmable switches that serves as the basis for our implementation. Then, we discuss NOX, an OpenFlow controller framework for writing applications for OpenFlow. Finally, we present an overview of prior work and their relevance to our work.

2.1.1 Access Control and Monitoring

Campus and enterprise networks are often large, heterogeneous, and difficult to manage, owing to network size, lack of good troubleshooting infrastructure and manual configuration. As such, network administration can be troublesome, manual, and error-prone. Network administrators often encounter situations where machines are infected or compromised. Today, the network operator must manually remove or quarantine the machine from the network, which is tedious. The network should offer flexible control over network traffic and also scale to a large number of users and traffic flows. To the extent possible, network management should also be automated.

2.1.2 Overview of Existing System

Figure 1 shows the current START architecture [23], the authentication system deployed on the Georgia Tech campus. It is currently based on virtual LANs (VLANs) and VLAN Management Policy Server (VMPS) [26]. The START system supports the following functions:

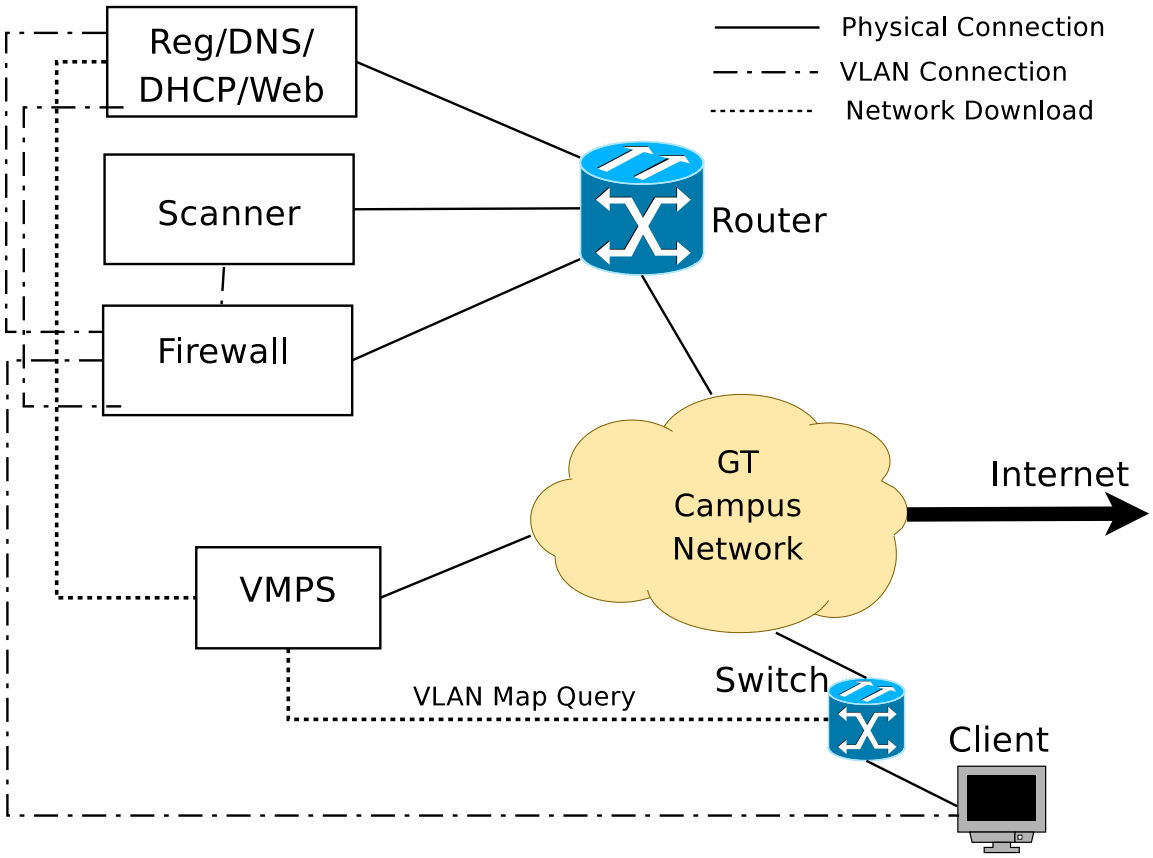

Figure 1: Current START Architecture.

Registration. The registration system provides the Web interface to the backend registration database, DHCP, DNS, authentication, and updates for external systems. The Web interface guides users through the registration process. The DNS server returns the IP address for the registration server for all DNS queries except for a list of domains needed for patching (*e.g.*, windowsupdate.com). The system runs two DHCP servers: One for the unregistered VLAN, and one for the registered VLAN. Each instance has its own configuration files that are created automatically from data in the registration system's database.

Scanning. During the registration process, the scanner device performs a host scan to detect any known vulnerabilities. If the scan reveals vulnerabilities, the user is presented with these vulnerabilities and asked to update the system. The firewall

allows traffic to the appropriate update servers.

Firewall. The registration VLAN uses a firewall to block network traffic to unregistered hosts. The firewall allows Web and secure Web (*i.e.*, port 80 and 443) traffic to pass so that hosts can reach update sites. The local switches determine the VLAN for each machine that joins the network. The switch will download VLAN maps periodically from a VMPS. Unknown MAC addresses are assigned to the unregistered VLAN and known MAC addresses are placed onto the appropriate subnet. VMPS periodically downloads the VLAN maps from the registration server. Security is enforced with ARP tables that map each MAC address to its registered IP address.

Several other mechanisms exist which try to solve the problem of network admission control. DNS-based solutions are common. The hosts are temporarily forwarded to the web-portal using DNS-based redirection. After authentication, the firewall waits for a timeout value before sending the correct DNS entry. This DNS poisoning technique may negatively affect post-authenticated internet use when the client machine references non-authentic data in its local resolver cache. Another approach could be to use normal switches and write a script to modify forwarding tables. Though this somewhat achieves the same effect as the OpenFlow protocol, yet there are some serious drawbacks with this approach. Since the method for manipulating the forwarding tables is not standard, different switches would require different scripts. Moreover, this approach is not very clean, as the scripts may not work with new switch firmwares which are not backward compatible. VLAN-based techniques offer more promise since hosts can be mapped to different VLANs which have different policies. In the next subsection, we focus mainly on the problems faced by START and VLAN-based architectures.

2.1.3 Problems with the current design

The current architecture has several shortcomings:

1. **Access control is too coarse-grained.** START deploys two different VLANs to separate infected or compromised machines from healthy machines. This segregation results in all compromised hosts residing on a single VLAN; such a configuration does not provide proper isolation, since these infected hosts are not isolated from each other. Additionally, relying on VLANs makes the system inflexible and less configurable, because VLANs typically map hosts to network segments according to MAC address, *not* according to individual flows.
2. **Hosts cannot be dynamically remapped to different portions of the network.** In the current configuration, when a machine is mapped to a different part of the network, it must be rebooted to ensure that it receives a public IP address, which is inconvenient because it relies on user intervention.
3. **Monitoring is not continuous.** Authentication and scanning only occur when a network device is initially introduced; if the device is subsequently compromised (or otherwise becomes the source of unwanted traffic), it cannot be dynamically remapped to the gardenwalled portion of the network.

Many of the current shortcomings result from the fact that security functions have been added to the existing network infrastructure after the fact. This design was natural when switches needed to be treated as "black boxes"; however, switch vendors have begun to expose a standard interface, OpenFlow [20], whereby an external controller can affect how a switch forwards traffic. We summarize OpenFlow below.

2.2 OpenFlow

OpenFlow-enabled switches expose an open protocol for programming the forwarding tables, also referred to as flow tables, and taking actions based on entries in these flow tables. The basic architecture consists of a *switch*, a centralized *controller*, and end hosts. The switch and the controller communicate over a secure channel using the OpenFlow control protocol [20], which can affect flow table entries on the switch.

Table 1: The header fields that an OpenFlow switch can map.

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

The OpenFlow protocol provides flow-level granularity for manipulating network traffic. A flow is defined as a 10-tuple including Ethernet, IP, VLAN headers as shown in Table 1. OpenFlow switches have three salient features: (1) A *Flow Table*, with an action associated with each flow entry, to tell the switch how to process the flow, (2) A *Secure Channel* that connects the switch to a remote control process (called the *controller*), allowing commands and packets to be sent between a controller and the switch using (3) The *OpenFlow Protocol*, which provides an open and standard way for a controller to communicate with a switch. When a new flow arrives at the switch, it forwards the first packet to the controller. The controller then sends an OpenFlow command to instruct the switch to install an entry in its flow table with an associated action to be taken when encountering future packets belonging to the same flow. Currently, all OpenFlow switches support three actions:

1. **Forward** this flow's packets to a given port or ports. This function allows packets to be forwarded.

2. **Encapsulate** and forward this flow's packets to a controller. In this case, the packet is delivered to a secure channel, where it is encapsulated and sent to a controller. This function may be used for the first packet in a flow, so a controller can decide if the flow should be added to the flow table.

3. **Drop** this flow's packets.

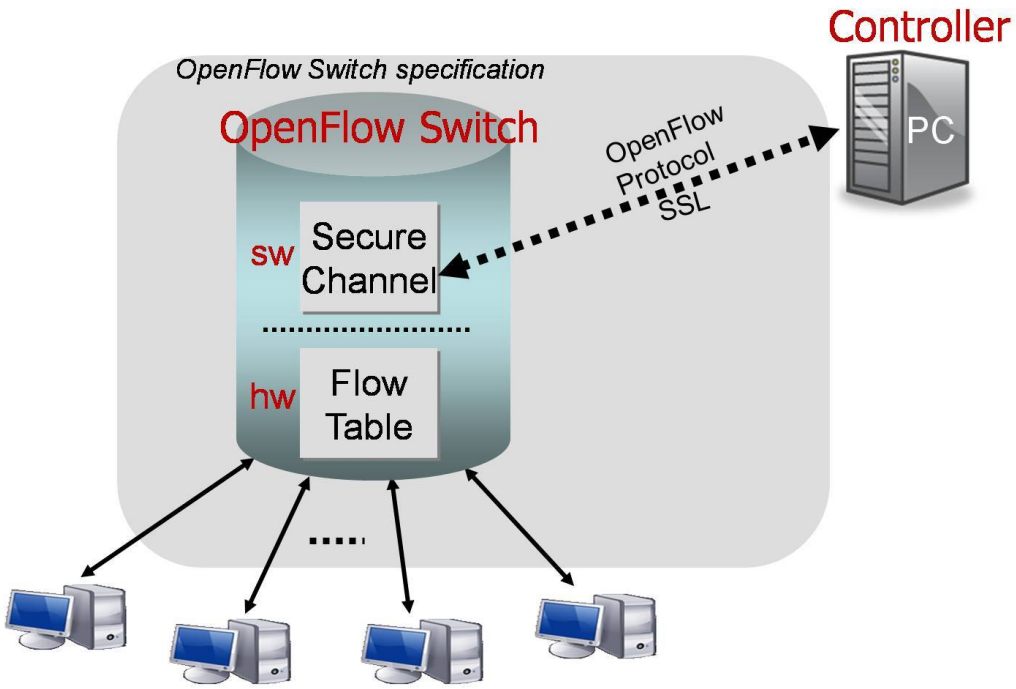

Figure 2: OpenFlow-enabled Switch. The Flow Table is programmed by a controller over the Secure Channel.

2.3 NOX: A controller framework

OpenFlow provides a platform for users to write their own controllers to control traffic through the switches. NOX [13] is a software platform that allows users to write their own custom controller components for OpenFlow. Figure 3 shows the NOX architecture. Components are essentially distinct *controller* instances on a single *network view*. NOX's network view includes the switch-level topology including all the network elements, users, hosts etc., along with network services. Programmatically, NOX applications are just a set of event handlers which are invoked whenever corresponding events are raised. Thus, the NOX framework serves as middleware between the OpenFlow switches and controller components. Primary components of a NOX-based network include a set of switches and one or more network-attached servers. NOX controls traffic at a per-flow granularity. In other words, once control is exerted on some packet, subsequent packets with the same header are treated in

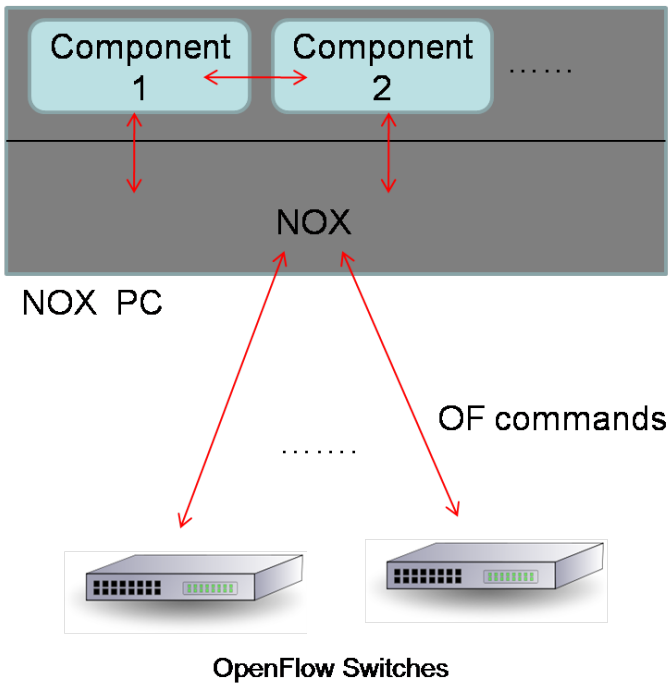

Figure 3: Components of a NOX-based network: OpenFlow (OF) switches and a server PC running NOX with different components. Communication among components and NOX happens using events and message passing.

the same way. To control the switches, NOX uses the OpenFlow protocol and installs flow-table entries on the appropriate switches as advised by the top-layer NOX components. The following subsection explains some of the programmatic interfaces available in NOX to create applications.

2.3.1 Programmatic Interface

NOX's programmatic interface revolves around events, a namespace, and the network view.

Events. To cope with changes in network events, NOX applications can register a set of handlers to execute when a particular event happens. Execution of these event handlers happen in the order of their priority. Events can be generated by OpenFlow switches themselves. Some of these include *switch join*, *switch leave*, *packet received*, *switch statistics received*, etc. Applications can also receive messages generated internally by NOX applications through the `post()` API while processing other messages.

We use this message-passing interface in Resonance for communication between main Resonance component and helper components.

Control. Management applications exert network control through the OpenFlow control protocol. The OpenFlow abstraction enables users applications to manipulate forwarding tables on switches. Applications can insert entries, delete entries, or read counters from entries in the flow table.

Network View and Namespace. NOX software comes pre-installed with a number of "base" applications that construct the network view and maintain a high-level namespace that can be used by other applications. These applications provide a mapping between high-level names and their bindings in the network view. Thus, application creators need only operate in high-level namespaces, not with low-level network bindings.

Higher-Level Services. NOX includes a set of system libraries to facilitate efficient implementations of common functions such as standard network services including DHCP and DNS.

We have chosen NOX as the base for our implementation. The flexibility and infrastructure offered by NOX made us choose it instead of developing our own custom controller from scratch. Moreover, NOX currently has a community of developers who are willing to provide support and debug issues. However, dealing with such a nascent piece of software does come with its drawbacks too. NOX lacks in documentation and most of the source of information comes through the source code itself and the NOX mailing list, but development is straightforward after the initial learning curve.

2.4 Related Work

The concept of dynamic, flexible access control goes back to task-based and Petri Net workflow-based methods [24], [17]. But, most of this work does not involve real system implementation or deployment. We have not seen any of these models applied

to network access control. Resonance is the first work of its kind with a design and implementation of a network access control model using programmable switches.

Resonance draws inspiration from 4D [11] and Ethane [3], both of which advocate controlling network switches from a separate, logically centralized system. Ethane [3] is perhaps the most closely related work to Resonance. Like Ethane, Resonance relies on a centralized controller. However, Ethane does not support continuous monitoring and inference-based policy control. Ethane focuses primarily on host authentication, as opposed to security-related problems such as monitoring and containment. Resonance extends the Ethane paradigm by exploring how *dynamic* security policies and actions (*e.g.*, actions based on alerts from distributed detection systems) could be more directly integrated into the network fabric.

As discussed in previous section, our implementation is based on the NOX controller framework. Recently, several NOX-based controllers have emerged. SNAC [22], is one such controller which provides a web interface to NOX to specify policies. SNAC is based on FSL [14], a policy language for NOX that allows network operators to write and maintain policies efficiently. Resonance focuses more on creating the actual policies that relate to dynamic access control and monitoring in enterprise networks. SNAC also provides an option to setup a web authentication portal. However, the portal is primitive and provides only basic authentication. There is no capability to dynamically change policies based on network inputs.

Resonance allows network devices to operate on the granularity of flows using the OpenFlow standard [20] and has origins in the designs of earlier protocols (*e.g.*, ATM [19]) and programmable switch architectures [25]. Recent trends in packet forwarding architectures (*e.g.*, [4]) have tried to achieve a similar shift towards the lower layers by having the software part of the switch make forwarding and pass it on to the hardware.

Some of the features of Resonance can be implemented using today's protocols.

VMPS [26] allows a network to map a host to its corresponding VLAN based on its MAC address. However, network operators achieve this mapping via manual configuration; if a host needs to be re-mapped based on a change in its state (*e.g.*, if the host becomes compromised), VMPS provides no mechanism for automatically remapping such a host; this remapping must either be done manually, or a higher-layer, on-path security middlebox must take appropriate action. Resonance's access control is both more dynamic and more fine-grained than the access control enabled by VMPS and VLANs.

CHAPTER III

RESONANCE: DESIGN AND USE CASES

Access control is a well-known paradigm for enforcing and implementing network security in campus and enterprise networks. In this chapter, we describe the design of Resonance and explain how it overcomes issues in current architecture. Resonance is based on a formal state machine access control model. We modified the formal model to simplify our design and make it more applicable to the problem at hand. We, then, present some of the use cases of Resonance, including admission control and rate limiting.

3.1 Overview

Resonance is a state-machine based access control framework which can solve wide variety of problems. Resonance works by specifying a state machine to represent network dynamics and associating each state with a set of access control policies.

Resonance works as follows:

1. Associate hosts with generic states and security classes.

2. Devise a state machine to capture network dynamics of hosts. In other words, the state machine should express changes in network policies over time based on events.

3. Install or modify forwarding state in switches based on the state of the host and related policies.

Next, we present a formal state machine model which can be used to prove security properties of a system. Resonance is based on a similar model, but with a few major

differences.

3.2 Formal State Machine Model for Access Control

Computation as a state machine model was first presented by D. E. Bell and L.J. LaPadula in 1976 [2]. The Bell-LaPadula (BLP) model is more generic than just an access control model; information flow can also be modeled using the BLP model. This model has been instrumental in building secure systems esp. where the model needs to be verifiable. The BLP model is abstract without any specifics of what subjects, objects, access rights, etc. mean in the real world. We architected Resonance by redefining states, subjects, objects and other entities in the context of network access control. Because of its closeness to express end applications, the BLP model is a good fit for our problem, even though real applications don't necessitate the need for mathematically sound model. In fact, a formal base serves as an additional feature of our design.

The finite-state machine model views a computer system as a finite set of states, together with a transition function to determine what the next state will be, based on the current state and the current value of the input. The transition function may also determine an output value. Transitions are viewed as occurring instantaneously in this model; therefore certain potential information channels (*e.g.*, those related to observing the time spent in a certain state) in real systems tend to be hidden by it.

The state machine model represents the system in terms of the triple (S, I, F), where S is a set of states, I is a set of possible inputs, and F is the transition function to move the system from one state to another. Security properties are proved using induction. Typically, assuming the initial state is secure, proving that each of the transitions lead to a secure state, the security of the entire system is guaranteed.

The BLP model is based on the following sets:

- *S subjects*

- *O objects*
- *A access modes* such as **read**(**r**) and **write**(**w**)
- *L security levels*

In the context of networks, subjects could represent hosts, objects could be network services and access models could be simple allow or deny rules.

System, states, and state transitions. A system can be thought of as an instantiation of a state machine with states *V*. Each state v is a member of the set of all possible states $V = (B \times F)$, where:

- B is the set of all possible *current access sets*. An element b of the current access set is a triple (s, o, a) - subject, object, access mode. It represents the accesses that are currently held by the subjects on the objects.
- F is a subset of $\mathrm{L}^S \times \mathrm{L}^O$. Each element of F is a pair of functions (f_S, f_O), which gives the security level associated with each subject and each object. This can be used to deduce relative security properties of subjects and objects, and the state as a whole.

A state transition is defined in terms of an *action* tuple (r, d, v^*, v). Here, r is an element of the set of *requests R,* and d is an element of the set of *Decisions, D.* A rule in the state transition is a function that associates (request, state) with a pair (decision, state). Essentially, rules correspond to system operations such as **get-read** (which alters the current access set), **give-access** {which alters the access matrix}, etc. The action tuple represents a transition from state v to v^* on request r resulting in decision d. The inputs to the system are a set of *requests R*, and the outputs are *decisions D.*

Security Properties. A secure state is defined by three important properties that are intended to express policies: the *simple security (ss-) property*, the **-property*, and the *discretionary security property*. The ss-property expresses clearance-classification policy. The *-property represents the policy of no unauthorized flow of information from a higher level to a lower one. These two properties represent mandatory access control. The discretionary security property reflects the principle of authorization and is expressed in an access matrix.

A state satisfies the ss-property if and only if, for each element of the current access set b, the security level of the subject dominates the security level of the object. The *-property is satisfied if and only if (1) for each **write** access in the current access set b, the level of the object equals the current level of the subject, and (2) for each **read** access in b, the level of the subject dominates the level of the object. The *-property ensures that, if a subject has read access to one object and write access to another, then the level of the first is dominated by the level of the second.

Informally, a system is defined as all sequences of actions with some initial state that satisfy a relation (represented by the set of possible action tuples) on the successive states. A state sequence z is secure if and only if each state in the sequence is secure. An element $(x,\ y,\ z)$ is called an appearance of the system, where x is a request sequence, y is decision sequence and z is a state sequence. An appearance is a *secure appearance* if and only if z is a secure sequence. Finally, a system is secure if and only if z is a secure sequence.

3.3 State machine based policy specification

To model real-time network applications, we investigated some of the underlying properties of a typical enterprise network. We observe that hosts are exposed to different kinds of traffic. Everyday, we see viruses, spam, malware, botnets appear. Hosts need to be constantly patched with new updates. Clearly, Enterprise networks are

driven by events resulting in constantly changing network state; hosts can become compromised, remain unauthenticated, be run by priority users for research experiments, open new network ports, users can come and go, links can go up and down. If we view the network as consisting of different elements and devices that dynamically change their behavior depending on network inputs, a state-machine based design for Resonance becomes natural.

Resonance has been designed in a different way than BLP to allow it to better describe existing systems. One appeal of BLP model is its abstractness, but the same quality makes it difficult to apply. It has no place for application-dependent security rules. The BLP model defines states as abstract relation "currently held" between subjects, objects and access methods. However, in case of network access control, having each flow represent a new state is overly restrictive and would explode the state address space. Thus, for Resonance, we define a state as representing in essence only the security levels of the subjects. Using this abstraction, we can actually define each state to be just a set of access control policies stating what network services are accessible to hosts. We also define subjects as being network hosts, objects being network services and access modes being whether or not the network service are allowed for those hosts. In the BLP model, the entire system is represented using a single state machine and related transitions. However, for Resonance, each host belongs to its own state, though still following the same state machine as other hosts in the network. The controller keeps track of the states of all the hosts in the network. As with the BLP model, we can argue about the security properties of the system by observing that a system is secure if all the transitions of hosts result in secure states.

In order to specify the state machine, we first identify the states. For example, in case of the admission control problem, a host can belong to one of several states. A host can be a new host which needs to authenticate. It can also be compromised

in which case, it has to be quarantined. On receiving inputs from network, the controller changes a host's state. In Resonance, a state represents a set of access control policy, allowing or disallowing specific traffic. Clearly, state machine is application dependent. We shall apply the Resonance technique to different use cases.

As with any other method, the design of Resonance can be best understood by considering different use cases and examples. In the next few sections, we discuss two different use cases for Resonance. As a first use case, we apply our state-machine model to the network admission control problem at the Georgia Tech Campus. In the second use case, we study how Resonance can be used to dynamically change rate limiting policies.

3.4 Network Admission Control: Redesigning START

Network Admission Control refers to the problem of filtering hosts which are allowed access to the network services. Users start by connecting their device to the network. Typically, they are presented with a web site to enter their authentication credentials. During this time, the network device is denied access to any traffic other than the web traffic. Once users verify their login information, they can continue surfing the Internet normally. This is a simplistic model of Network Admission Control problem we have described so far. As we introduce new network elements, *e.g.* firewalls, security middle-boxes etc., network policies might become complicated. As an instance, to keep network free from botnets and viruses, a network scanner can be added to the network to monitor botnet activities. Implementing policies which can isolate hosts, discovered as being part of a botnet by the monitor, on the fly can be tricky.

START is the technology currently used in Georgia Tech dormitories to tackle this problem. It allows users to connect their machine to the Georgia Tech network. The users are then presented with a captive web portal. After authenticating with their credentials and a basic machine scan, the users get access to the Internet. As discussed

in Section 2.1.2, START suffers from VLAN management issues and lack of continuous network monitoring. Resonance tries to overcome these problems by mapping this problem to a state machine model. We examine some of the inherent properties in the START system and infer states directly from the way START architecture works.

3.4.1 State Description

Based on the intrinsic nature of "Network Admission Control" problem and the START design, we have identified the following 4 states:

1. **Registration.** In this state, the user is not allowed access to any network service. All web traffic is redirected to the web portal, where the user is presented with a web site. From this state, a user can move to Registration state once the authentication succeeds.

2. **Scan.** In START, after the user authenticates, it has to be scanned for potential vulnerabilities. We introduce Scan state that corresponds to the machine being scanned. The host is allowed access to only updates sites, e.g. Microsoft update websites. The hosts can get the required patches before they connect to the Internet. This state can lead to either Quarantined state or Operation state.

3. **Operation.** In this state, users are allowed full access to the Internet.

4. **Quarantined.** Hosts belonging to this state are denied access to all network traffic. The host OS needs to be reinstalled or the infection manually removed before the host can be moved back to the registration state.

The entire set of policies governing each state is shown in Table 2 and Table 3. The "Actions" column corresponds to OpenFlow commands to be executed when the switch encounters packets which match the corresponding "Match" column. For example, in "Registration" state, all packets matching "ARP" (Ethernet frame type 0x806) are flooded on output ports by the switch.

Table 2: Flow table entries for Registration and Operation states.

Registration State	
Match	**Action**
Ethernet Type ARP (0x806)	FLOOD
UDP src_port=68 dst_port=67 (for DHCP)	FLOOD
UDP src_port=67 dst_port=68 (for DHCP)	FLOOD
UDP src_port=53 dst_port=* (for DNS)	FLOOD
UDP src_port=* dst_port=53 (for DNS)	FLOOD
TCP Dst port 80/443/8080	REDIRECT (to web portal)[1]
*	DROP

Operation State	
Match	**Action**
*	FORWARD/DROP (according to policy lookup)

3.4.2 State Transitions

Figure 4 shows all the state transitions for Resonance as applied to START.

We summarize the transitions as follows:

- **Registration State Transitions.** Transitions are possible to Scan state and Quarantined state. Registration state to Scan state transition happens on receiving input from web portal. Transition to Quarantined state occurs after a user incorrectly authenticates for maximum allowed times.

- **Scan State Transitions.** If the scan on the host was clean and the patches successfully applied, then the host can transition from Scan state to Operation state, else the host is moved to Quarantined state.

[1]Note that REDIRECT is not an actual command specified in OpenFlow spec. We use it to denote a set of flow entries which direct traffic to the appropriate host.

Table 3: Flow table entries for Scan and Quarantined states.

Scan State	
Match	**Action**
Ethernet Type ARP (0x806)	FLOOD
Dst_IP=(Scanner's IP)	FORWARD
Src_IP=(Scanner's IP)	FORWARD
TCP dst_IP=update sites	FORWARD
TCP src_IP=update sites	FORWARD
UDP dst_port=53 (DNS port)	FORWARD
UDP src_port=53 (DNS port)	FORWARD
*	DROP
Quarantined State	
Match	**Action**
TCP dst_port=80/443/8080	REDIRECT (to quarantine web page)
*	DROP

- **Operation State Transitions.** On getting input from the distributed IDS, a host can be moved back to the Scan state.

- **Quarantined State Transitions.** The controller can get input from the network administrator to allow the host to move back to the Registration state.

3.4.3 Resonance Step-by-Step

In this section, we explain how Resonance works step-by-step when a host connects to the network. We also describe how the controller manages the flow-table entries when two hosts attempt to communicate. Finally, we explain how a machine changes states and how the controller changes the flow-table entries accordingly.

Let us consider a simple setup with four OpenFlow switches, one controller, two hosts, and four servers, as shown in Figure 5. When OpenFlow switches establish a connection with the controller using a secure channel, the controller installs default flow rules for DNS, DHCP and ARP traffic with FLOOD action as indicated in

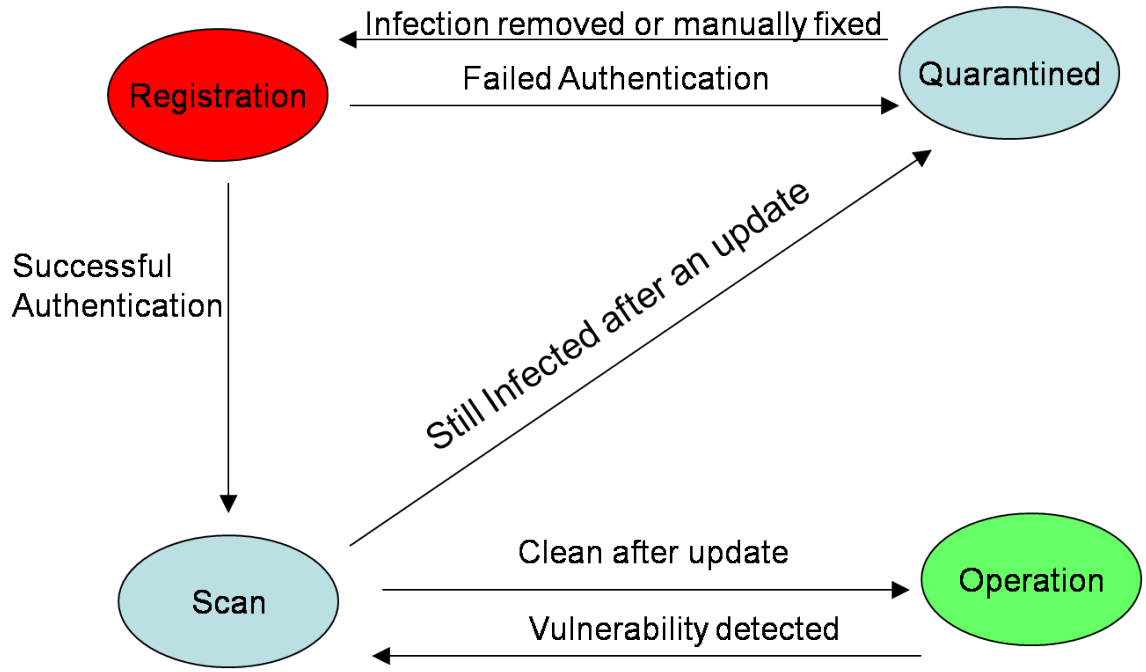

Figure 4: State transitions for a host. The controller tracks the state of each host and updates the current state according to inputs from external sources (*e.g.*, network monitors).

Table 2. When a new host is introduced on the network, it first broadcasts a DHCP discover message. Since the switches have already been configured to allow all DHCP to be flooded, the host is able to acquire a network IP address following the standard DHCP protocol. The controller also adds the host to its database of hosts and marks its state as "Registration". Quarantined hosts may still be able to reach the Internet by tunneling traffic over DNS. We are working on solutions that can avoid such communication.

In the Registration state, if a host initiates any traffic that is not DHCP or ARP, the controller installs a new flow-table entry into the switch with action="DROP", unless the traffic is HTTP, in which case the controller installs an entry to redirect traffic to the portal, which redirects the user to the authentication Web site. A machine in the Registration or Quarantined state cannot initiate a connection, but it can always receive packets from a machine in the Operation state. We make this

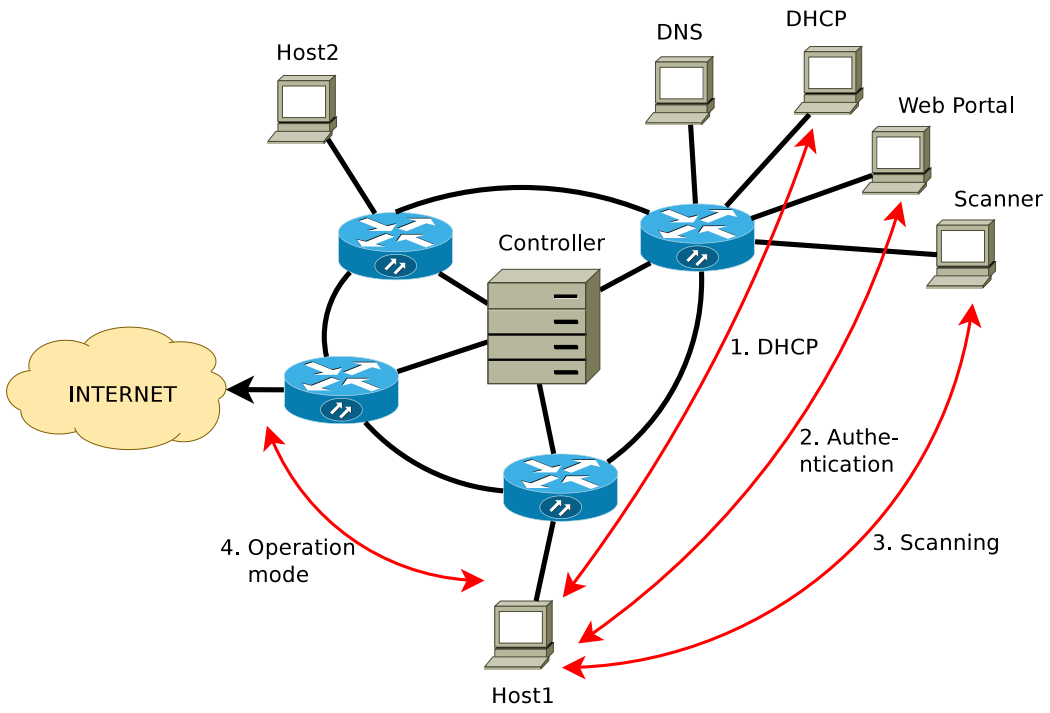

Figure 5: Applying Resonance to START.

policy rule for simplicity.[2]

The Web portal allows the user to authenticate and notifies the controller of the status of the authentication via a separate connection. Upon successful authentication, the controller moves the host to Scan or Quarantined state. It then deletes all old flow-table entries corresponding to the host's MAC address and installs a new set of flow-table entries, as shown in Table 3. The only change made from Registration state to Scan state is that the host can communicate with the scanner and update sites. The scanner scans the machine for potential vulnerabilities. If the machine is found to be vulnerable, it is redirected to update sites to patch those vulnerabilities. Once the update patches have been applied, the scanner notifies the controller, which transfers the host to the Operation state and updates the flow-table entries accordingly.

Once in the Operation state, the host can connect to any other Internet destination.

[2]An immediate implication is that communication is possible from Operation state machines to quarantined host machines. Because quarantined machines cannot initiate external communication, they are relatively immune to threats.

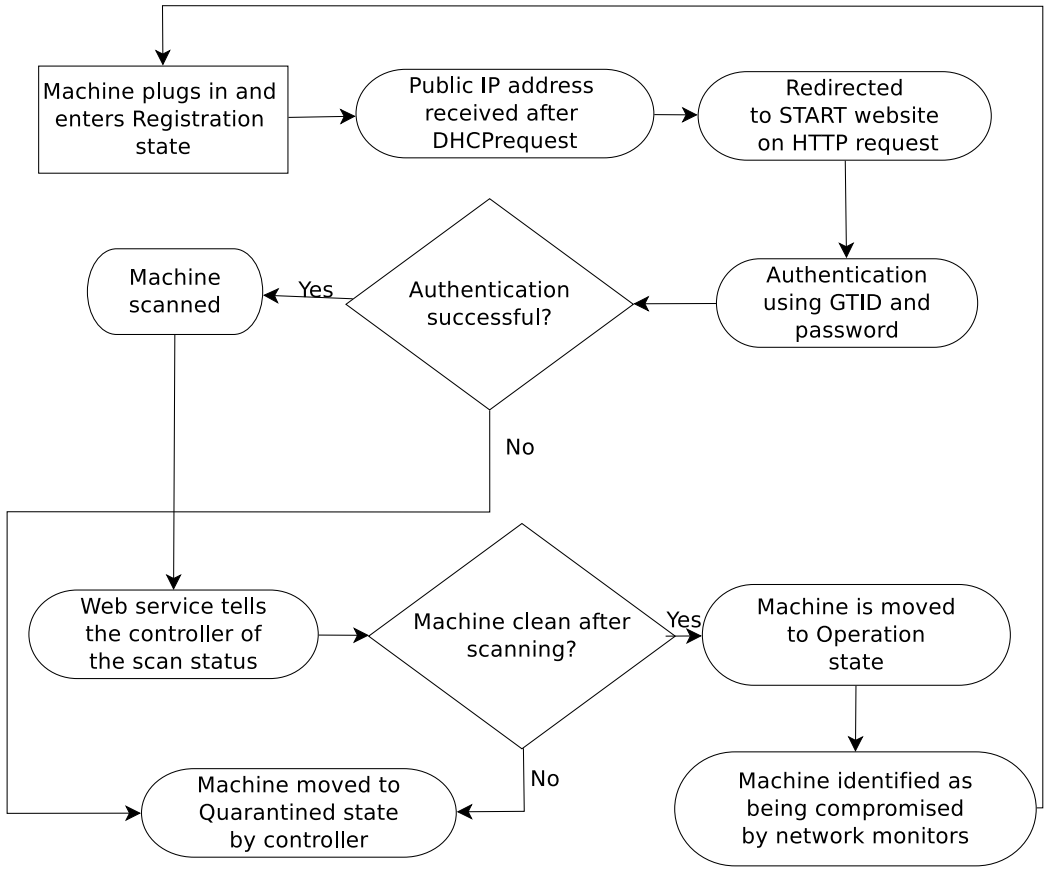

Figure 6: Flow chart representing sequence of actions in Resonance when applied to START.

During normal operation, a host may become compromised. If network alarms inform the controller about the event, the controller can then shift the host to the Scan state. Figure 6 shows a flowchart representing this entire procedure.

3.5 Dynamic Rate Limiting

Rate limiting is used to control the rate of traffic sent or received by hosts. Traffic that is less than or equal to the specified rate is sent, whereas traffic that exceeds the rate is dropped or delayed. Rate limiting is typically performed by policing (discarding excess packets), queuing (delaying packets in transit) or congestion control (manipulating the protocol's congestion mechanism). Policing and queuing can be applied to any network protocol. These can even be implemented in a Linux box

using simple iptables rules. Congestion control can only be applied to protocols with congestion control mechanisms, such as the transmission control protocol (TCP).

Resonance's state-machine based design can perform dynamic rate limiting by allowing traffic to flow through different rate limiters on the fly.

State Description. Enterprise business models characterize how different class of users and/or accounts should be allocated bandwidths. Other than using account information, we can also use time of day to restrict volume of traffic flow during peak times. Using this as the base, defining a state is as simple as categorizing the bandwidth allocations at specific time of day and the account type. In our example, we have defined four states corresponding to four possible combinations of the tuple $\langle$A, T$\rangle$, where A is the account type and T is the time of day.

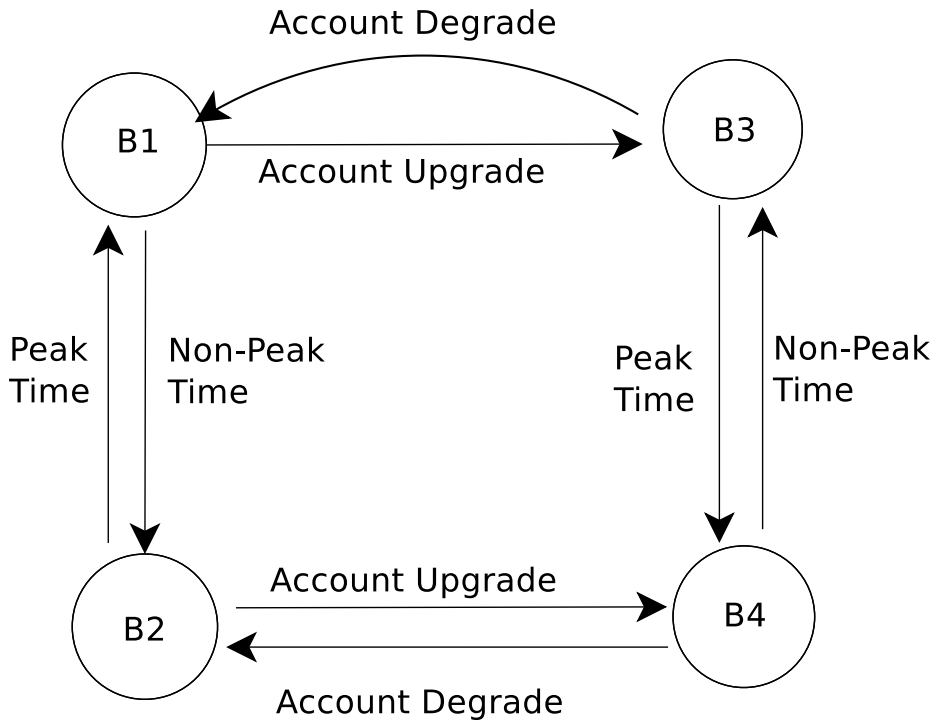

Figure 7: State transitions for dynamic rate limiting.

Inputs. There are two kinds of inputs allowed: R, request for account change and D, the current time of day. Accounts are either guest or premium. Premium account users get a higher share of bandwidth as compared to guest account users. Thus, R

contains two values, g and p. Information about account upgrades can be sent to the controller in the same way as for other inputs. Time of day can be divided into peak time and non-peak times. In general, high volume of traffic is observed around late afternoon and at other times of day network traffic is low. D contains the values p and n, for peak and non-peak traffic times.

State Transition Function. Figure 7 shows a simple transition function depicting how a flexible rate limiting policy can be implemented in Resonance.

There are transitions of two types, based on the input type. For account upgrades, the state is moved to high bandwidth. Similarly, the controller moves a host to a high bandwidth state if the time of day corresponds to low traffic time.

CHAPTER IV

IMPLEMENTATION, DEPLOYMENT AND PRELIMINARY EVALUATION

4.1 Implementation

We have implemented Resonance over NOX, a platform for creating new controller components. NOX provides basic interface to send and receive messages from the switches, and allows developers to create new components for interacting with the switch. Our current development uses NOX version 0.5.0 and OpenFlow version 0.8.9. In this section, we briefly outline machine configurations and setup, the organization of the Resonance code, discuss component interactions, show how we bootstrapped our initial setup, and give an overview of low-level interactions between Resonance and OpenFlow switches.

Network elements and configurations. All server machines are Dell PowerEdge 1900 machines running Ubuntu Linux 9.04 with 8GB RAM and Intel(R) Xeon(R) Quadcore 2.5GHz processors. There are six 1Gbps network interfaces attached to each of them. The Web portal currently runs Apache version 2.2.11 and PHP version 5.2.6. Configuring the web portal and making redirection to work required considerable effort. Our initial attempts at DNAT-based redirection using iptables [16] on promiscuous interface failed. Later, we found that iptables doesn't work directly on promiscuous mode. To solve this problem, we created a bridge and connected the physical interface to the bridge. The bridge allowed all packets to be captured and sent to iptables for DNAT processing. For IP address allocation and DNS, we have installed a unified cache-DNS and DHCP server using dnsmasq [6].

Base Resonance code. Most of the Resonance code is written in C++. The controller maintains a mapping between host machines and their corresponding ports on a particular datapath, which can be thought of as a learning switch. Each datapath represents a new connection from the switch to the host. When a host requests a new flow, the controller checks which state the host belongs to. If it is a new host, it adds its MAC address to the state machine list and marks its state as being in the Registration state. Thus, the controller maintains two mappings, one for maintaining the state machine and the other to find out which port the packet for that machine should be forwarded to.

For receiving inputs from external network elements, a separate communication path must exist. In Resonance, all external communication is handled through a separate component called Messenger. Messenger is a TCP server implemented as a separate component in NOX. It listens on TCP port 9999 and spawns a new thread for every new communication request it receives. Then, it forwards the received message to the Resonance component. In NOX, message passing between components is done through `post()` API. The received message from external communication is passed to the Resonance component using `Msg_event` structure in `post()`. We register a `Msg_event` handler in the Resonance component which handles messages received from the messenger. The event handler then updates the states of the host depending on the input from the web portal, scanner and the network monitors. Messenger-Resonance interactions are shown in Figure 8.

Bootstrapping. This refers to the procedure and configurations involved in booting up our setup after all physical connections are done. For this, we assume that all the switches know about the controller beforehand. There are discovery components already available in NOX, which can be used to do bootstrapping. In our implementation, we have manually configured the OpenFlow switches through CLI

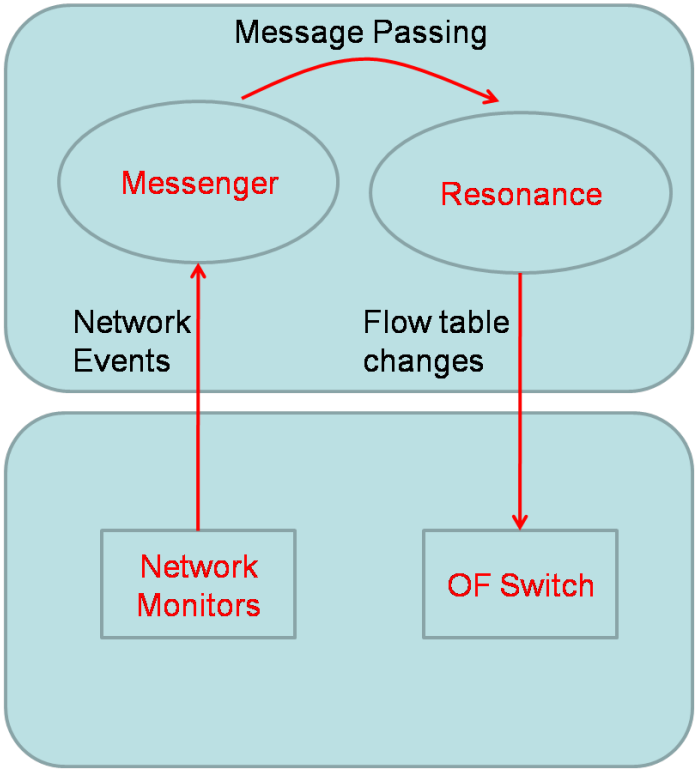

Figure 8: Event handling and message passing between Resonance and Messenger components.

(command-line interface) to connect to the controller using a hardcoded IP address and port number. We also assume that all the switches as well as the web portal, the scanner, and the network monitors are trusted. Just like the discovery component, authenticator components are already available in NOX to handle authentication for trusted devices.

When the switches initiate a connection to the controller, the controller immediately installs default wildcard flow rules for DHCP, DNS, and ARP. In our current implementation, we flood DHCP, DNS, and ARP requests over the entire network. All flow table entries, other than DHCP, DNS, and ARP are exact match entries and are handled as and when new flows arrive. For example, for a host in Quarantine state, the first packet for any new network session will be sent to the controller. Knowing that the host is quarantined, the controller sends a flow table entry to the

switch to drop subsequent packets for that flow.

A minor implementation question could be: Why not set all flow policies for a particular state in the switch beforehand instead of doing it on-demand? It is possible to do so using wildcard feature available in OpenFlow switches. But, these rules need to be on a per host basis, resulting in flow table size being equal to the number of hosts in that state. Currently, number of wildcard entries are restricted to around 100 in most of today's switches (because of hardware limitations of TCAMs), which makes this approach infeasible. In addition, wildcard matching is handled in software which can lead to significant performance hits. From the experiments we have performed, we see that the maximum number of flows come from DNS, DHCP and ARP requests. So, we use wildcard entries for such flows instead, since querying the controller for such flows can overload the controller and incur delay penalties.

Operation. Whenever the controller receives updates from the Web portal and other devices, it must change the state of the corresponding host. It does so by deleting all the current flow table entries for that host by sending an OpenFlow message with the OFPFC_DELETE command and moving the host from one state to the other depending on the input received. Individual flows are setup as and when they are received by the controller. Because all flow-table entries are deleted whenever there is a state change for a host, when the host tries to establish a new flow, a new message is sent from the switch to the controller. The controller then takes appropriate action, either allowing (using FORWARD command) or blocking (no action field in OF message) flow, depending on the current state of the host.

4.2 Deployment

The campus network currently supported by START was recently upgraded to include approximately 275 switches that are capable of supporting the OpenFlow firmware. One of the more significant practical challenges in the campus deployment will be

straining the scalability of the system on a production network without disrupting connectivity. For example, the proposed architecture may involve installing many flow table entries in the switches, which may either exhaust memory or slow lookup performance if entries are not stored efficiently, or if state is not offloaded to the controller. To address this concern, we will first stress-test the design on the research testbed and subsequently the architecture on a smaller number of production switches before completely rolling out the architecture.

We have successfully been able to deploy Resonance in three buildings at Georgia Tech Campus. Figure 9 shows the current status of our deployment. The deployment is a dedicated network that is physically separate from the production network and yet has its own IP prefix and upstream connectivity. This platform allows us to develop and test Resonance before deploying it on the production network. We have both wireless and wired connectivity on the testbed. Our current setup spans three buildings at Georgia Tech: Technology Square Research Building (TSRB), Klaus Advanced Computing Building (KACB) and College of Computing Building (CCB). We have OpenFlow-enabled 48×1G switches from three different vendors: NEC, Toroki, and HP. Switches from Toroki are LB4G Quanta models with OpenFlow firmware version 0.8.9. NEC and HP switches both have 0.8.9 versions of the OpenFlow firmware. There is an access point in the networking lab to support wireless communication. As of now, the IP address space is taken from BGP-Mux setup [9]. We plan to expand our IP space using the already allocated IP address space from RNOC.

4.2.1 Connectivity and Setup

All buildings are connected to each other using the NEC switch in CCB. Fiber paths connect both TSRB and KACB directly to CCB. Currently, all users connect to the Resonance network through the KACB building. In KACB, the OpenFlow-enabled

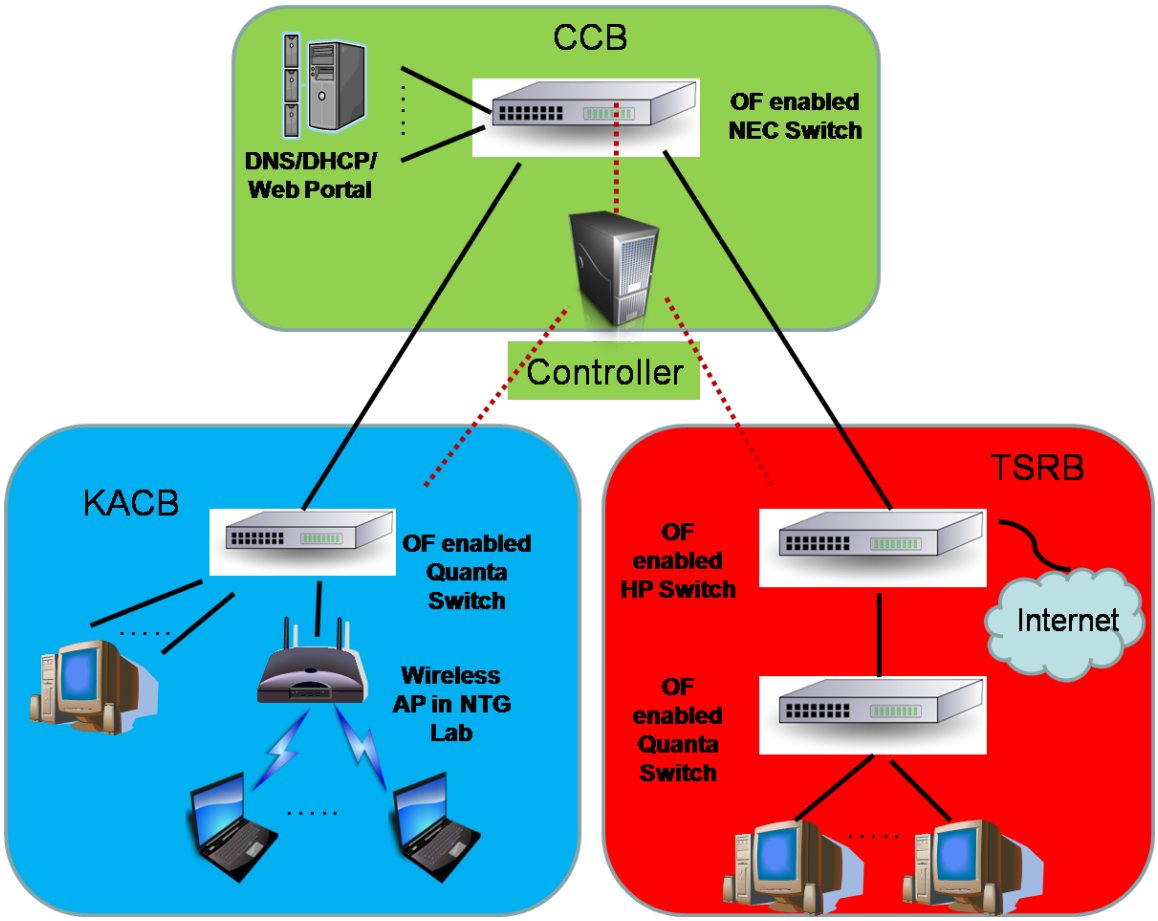

Figure 9: Current research testbed with connectivities between three campus buildings.

Quanta switch is present in the data closet in the third floor. There are two connections coming out of the switch; one to the NEC switch and the other to the controller, both in CCB. Other ports on the Quanta switch are directly patched to ports in the networking lab. These ports in networking lab serve as entry points to the Resonance network. A wireless access point is connected to one of them and facilitates wireless connection in the lab.

Since the Quanta switches only support out-of-band configuration, we need a separate control path to the controller from the switch. For this purpose, a separate virtual LAN (VLAN) has been assigned from KACB to CCB, and from TSRB to CCB. Controller in CCB and Quanta switches in KACB and TSRB are part of this

VLAN. All control traffic including initial OpenFlow handshakes between the switches and controller take place over this VLAN. Although out-of-band configuration allows better segregation of control and data traffic, yet for actual deployments such a configuration becomes infeasible since it necessitates the existence of separate VLAN paths from all switches in the entire network to the controller. On the CCB side, we have placed all management machines, including Web Portal machine, Controller machine and the DHCP/DNS server machine in the CCB machine room. All these management devices are connected directly to NEC switch in same rack. Connectivity with TSRB is still not fully functional. We have HP and Quanta switches in TSRB which are yet to be connected to controller in CCB. The RNOC switch, the gateway to the Internet for RNOC IP subnet, is also situated in TSRB, is waiting to be put online on the Resonance network.

Accessing the Resonance network. Users can directly log in to the Resonance network using WEP key for the access point located in the networking lab. On accessing Web, the users will be directed to the Web Portal where they have to authenticate using a hardcoded username and password. At this stage, the users get access to the Internet. We plan to introduce a IPS/IDS detection device and a scanner on this test network. The controller can use inputs from these middleboxes to complete all the state machine transitions.

A small working version of the Resonance system was demonstrated at the GENI Engineering Conference (GEC7) [10]. We explain the demonstration setup, along with screenshots in more detail in Appendix A. While being used for testing and developing Resonance, existing infrastructure can also be leveraged by other research projects for their own deployment, which can run side-by-side. This can be made possible by using FlowVisor [1] on top of NOX. FlowVisor provides the ability to direct different control traffic to different controllers.

4.3 Preliminary Evaluation

Performance and scalability of Resonance are as important as its functionality itself. In this section, we tackle the issue of whether Resonance is scalable enough to handle an enterprise network and has a reasonable performance to be in use. We evaluate two aspects of Resonance. First, we investigate scalability of Resonance in terms of whether it can handle the bulk of traffic generated by the network. The number of entries in the flow table of OpenFlow-enabled switches should never cross the hardware limit. Second, Resonance should not introduce unacceptable delay.

First, we must be aware of OpenFlow-enabled switch's capability. According to the specification, OpenFlow software switch can support 131,072 exact match entries and 100 wildcard entries. There are several OpenFlow-enabled switches manufactured by vendors including Toroki and NEC which currently provide around 1500 entries in the flow-table. We expect a product to come out with more capabilities in the future.

To simulate traffic that is similar to the real world, we captured traces from the Georgia Tech campus network. Georgia Tech maintains a network which represents a typical network infrastructure that many campuses and enterprise networks implement.

4.3.1 Flow table size analysis

We assume a single switch will roughly handle a /16 subnet. Here, we focus on two different /16 subnets, A and B. An hour of sample traffic is captured on a server listening to a certain part of the campus network on a typical weekday from 2pm to 3pm. It is then classified based on source and destination IP address matching to a certain /16 subnet. Each traffic is analyzed to extract the number of concurrent unique flows in a specific time interval. With this information, We measure how many flow-table entries are needed to manage the traffic, and whether an OpenFlow switch is able to handle it.

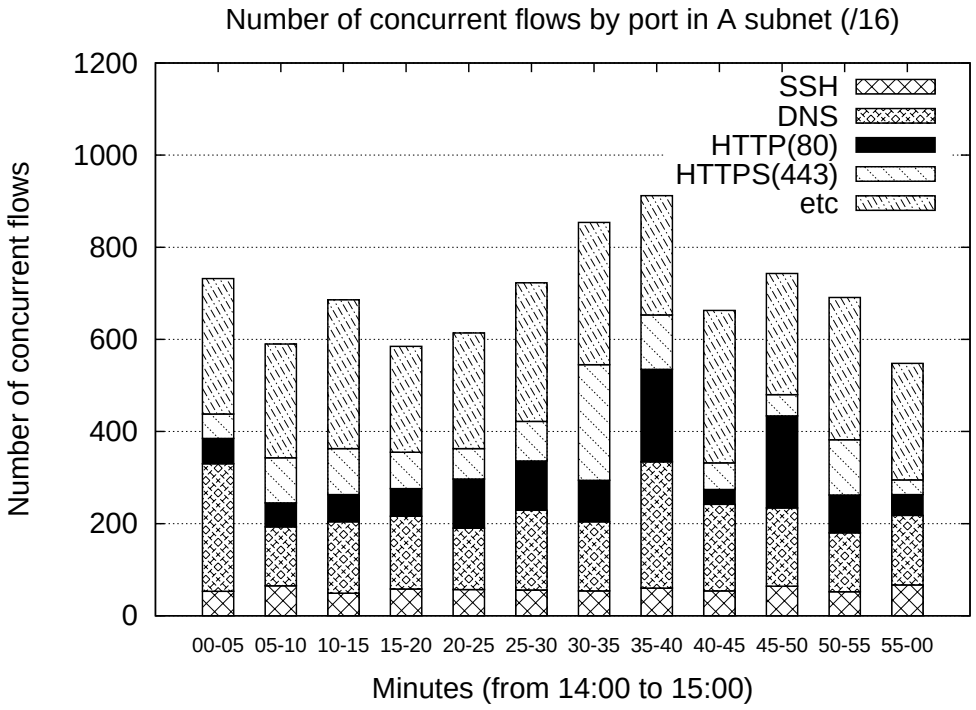

Figure 10: Number of concurrent flows in subnet A.

Figure 10 and Figure 11 show the number of unique flows in a given 5 minute interval along an hour. Each unique flow will map to a single flow-table entry in a switch. This can be used to predict how many flow-table entries are required to handle the network traffic managed by the subnet. As shown in both subnets, the number is mostly below the 131,072 threshold. In addition, there will be multiple flows which can be dealt with several wildcard entries. If optimized by using wildcard entries effectively, the numbers can be decreased further. Better timeout values for each entry based on their protocol and service will help as well. The graph promises that in a typical campus network, Resonance is scalable in terms of flow-table entries generated in each switch. As OpenFlow switches will only be deployed at the edge network, the scalability can be achieved as far as the switch is responsible of a reasonable subnet

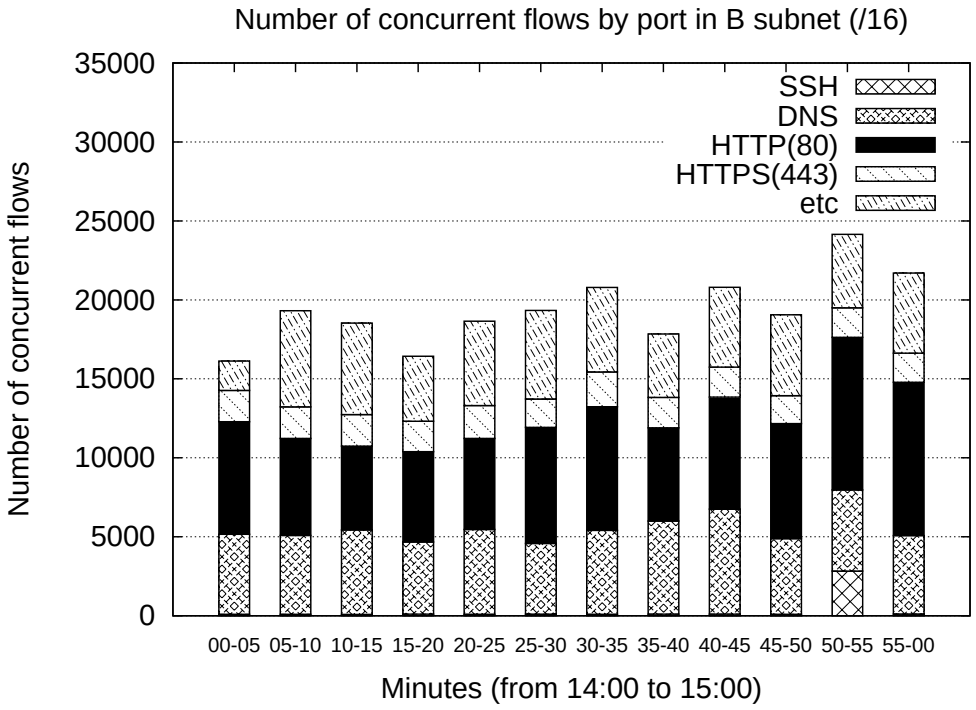

Figure 11: Number of concurrent flows in subnet B.

size and number of hosts.

4.3.2 Flow setup and performance experiments

Resonance should not impose noticeable delay in the network itself. In this section, we show the evaluation result of Resonance in terms of flow setup time and packet transfer delay.

In OpenFlow framework, there is a delay introduced by consulting the central controller. The controller is responsible of sending a response and installing flow table entries in the requesting switch. Flow setup time is the delay caused by setting up an entry in the switch's flow table. In other words, flow setup time is the difference between the time a switch sent a request to the controller, and the time it received a response. As Resonance has multiple steps of deciding what to do for each new flow,

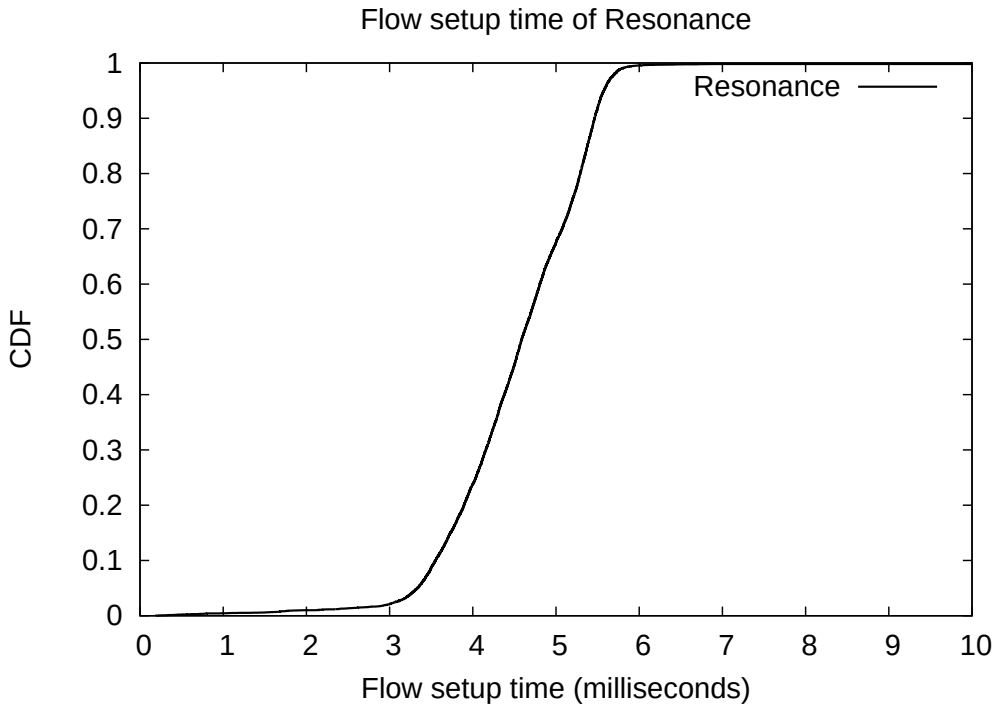

Figure 12: Flow setup time delay using Resonance.

the flow setup delay is an issue. The delay caused by Resonance decision making process should be in a tolerable boundary. To measure this delay, OpenFlow-related traffic is collected at the switch, and analyzed according to the OpenFlow protocol to find the request and response packet. Then, the time difference is calculated to measure the flow setup time using the timestamp value from these packets.

Figure 12 shows the CDF of the time spent on flow setup. The graph shows that the flow setup delay is mostly below 6 milliseconds, where the majority is between 3 to 6 ms. This is a negligible amount of time compared to other delays caused by the network itself. In addition, this delay only occurs at the first time the flow table entry is inserted. There will be no further delay when incoming flows matches an entry in the flow table, as far as the entry does not time out.

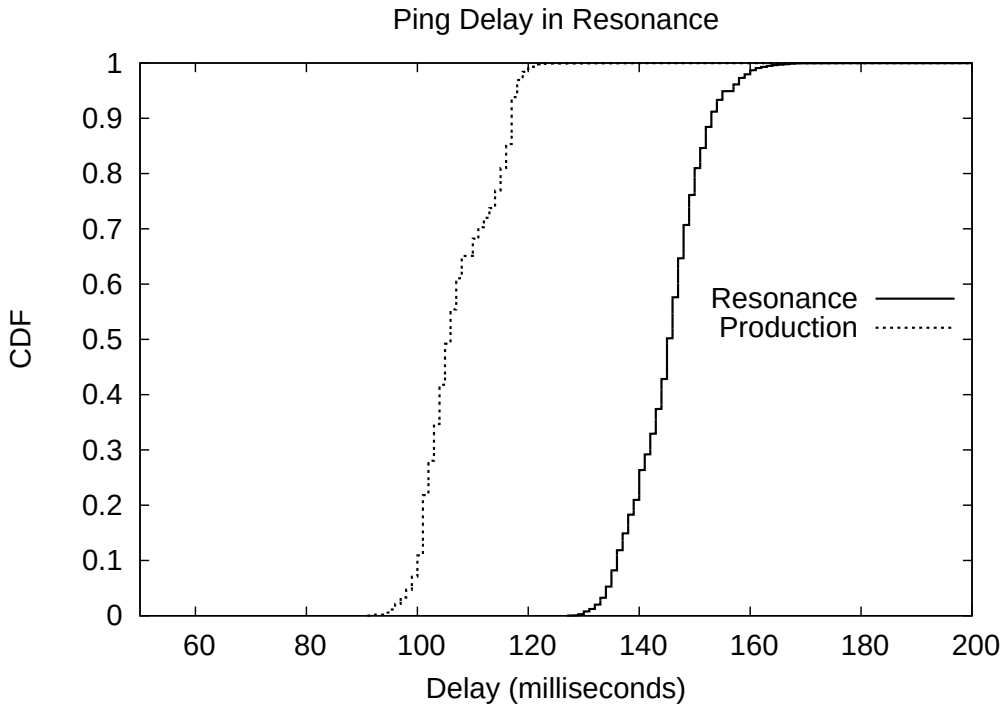

Figure 13: Ping delay in Resonance.

A straightforward way to evaluate the performance of Resonance is to measure a ping delay. Figure 13 shows the delay in Resonance compared to the existing network. As Resonance has much more functionality implemented within it, more delay is an expected behavior. The difference between production and Resonance is around 40 ms. One of the reasons of this additional delay is that these experiments were performed on Linux kernel reference implementation switches. OpenFlow software switches are installed in a normal Unbuntu Linux box, and it is bound to be slower than normal hardware switches because traffic has to be processed up to the application layer, traversing the entire network stack for processing. If an OpenFlow-enabled hardware switch is used instead, so that the processing can be done in the hardware

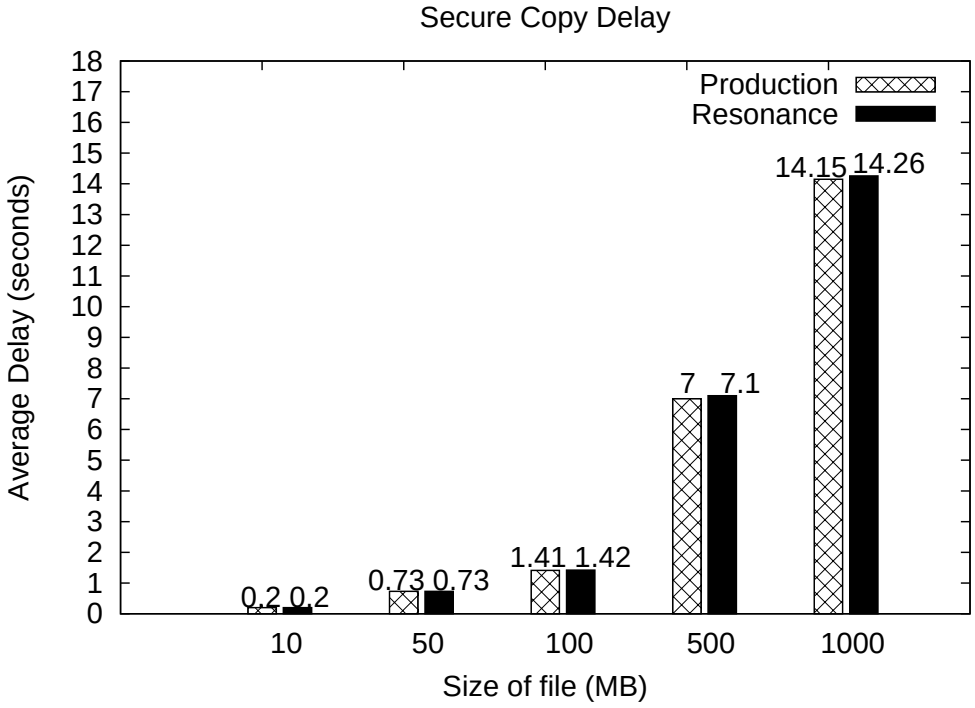

Figure 14: Secure copy delay in Resonance.

layer, the difference is expected to be much less. The flow setup delay will not contribute much to the ping delay, as it only has influence at the first time the flow enters the switch. Another reason can be the delay caused by searching the flow table for a match. However, this will be insignificant, as OpenFlow uses an effective hashing algorithm to perform this task.

In addition to the delay in ICMP, measuring the delay with an application using TCP will be more realistic. Therefore, we measured delay when using secure copy. The files being transfered range from 1 MB to 1 GB. As shown in Figure 14, there is no noticeable delay using Resonance compared to the production network in the scale of seconds. The difference will be more apparent as the file size increases, but it was not significant up to 1 GB.

4.4 Challenges

Despite the promise of Resonance's design and recent technology trends that could make its deployment more feasible, we continue to grapple with many challenges in our test deployment.

- **Scale** When deploying the architecture on the campus network, we expect to encounter numerous challenges involving scalability. For example, the Georgia Tech residential network must support approximately 16,000 users; the portion of the campus that runs START comprises more than 13,000 network ports, and future plans include expanding START to more than 40,000 active ports across academic buildings and merging START with the (separate) access control system currently used for the campus wireless network. A significant challenge will be implementing dynamic, fine-grained policies with flow-table entries, without exhausting switch memory or slowing forwarding. Though our initial investigations in Section 4.3 show positive signs about the scalability of Resonance, yet we believe that as we carry out our deployment efforts more aggressively, we shall encounter unforeseen hurdles in terms of traffic handling and flow capacity limits on switches and controller. To cope with these issues, we can definitely segregate different parts of the network, handling only a subset of hosts based on the hash value of their MAC addresses. The underlying mechanism behind Resonance permits several controllers to co-exist without the need for much co-ordination between them. FlowVisor can also be used to slice different network subnets to reduce the burden of traffic handled by the controller. Recent proposals for optimizing customizable forwarding [4] may offer a useful starting point too.

- **Responsiveness** End hosts and network devices must be able to quickly authenticate to the network controller; similarly, the network must be able to

quickly quarantine a compromised host and curtail unwanted traffic. The current design is inadequate in this regard, as it has a single VLAN for quarantined hosts and requires hosts to re-boot to reassign hosts from one VLAN to another. The Resonance architecture offers more fine-grained, dynamic control over hosts' traffic, but the control framework between the switches and controller must still be able to map hosts from one part of the network to another as quickly as possible. To enable this responsiveness, distributed inference must be fast, and the controller must be able to quickly and reliably alter the behavior of the switches themselves.

- **Integration with monitoring** The current START network access control system scans hosts when they are first introduced into the network but cannot re-assign these hosts to different networks when they are deemed to be compromised. In our ongoing work, we already have a setup in place that can leverage today's state-of-the-art IPS/IDS systems to classify and quarantine compromised hosts.

- **Securing the control framework** The effectiveness of Resonance depends on the existence of a secure, reliable channel between the controller and the switches. The control messages between the controller and the switches must be authenticated (so that switches do not alter their behavior based on arbitrary control messages), and the channel must remain reliable and available, even when network utilization is high or the network itself comes under attack.

CHAPTER V

CONCLUSION

Existing enterprise networks leave network monitoring and access control to higher layers (*e.g.*, DHCP, application-level intrusion detection, etc.) and place considerable amounts of trust and responsibility into the network devices themselves, resulting in complex, error-prone configurations for enforcing security policies. To remedy these ills, network access control must be more dynamic and fine-grained, and it must make as few assumptions as possible about the behavior of the host. We have introduced a new framework, Resonance, for specifying dynamic access control policies for networks, described how this might be implemented in an OpenFlow-based architecture, and shown how to apply Resonance in the context of the Georgia Tech's network access control framework. In addition to the test and operational deployments themselves, we are exploring how Resonance can support more complex access control policies.

Future Work. As next steps, we plan to aggressively pursue our deployment efforts to include more switches, buildings and hosts. Challenges continue to shape Resonance's design and development. We have addressed some of them in this thesis. We continue to invest our efforts towards making our system scalable and robust, giving near-production performance, and making it free of security threats. As of now, our wireless infrastructure makes use of OpenFlow switches as a back-end. OpenFlow-enabled access points would add more diversity to our testbed. We also plan on adding better front-end to monitor traffic, troubleshoot issues and simplify usage. We are exploring further avenues for Resonance's application and use cases, *e.g.* QoS, load balancing, IP/Interactive TV provisioning, etc. Our end-goal is to replace existing

security solutions in Enterprise networks with Resonance, making network security and management flexible, dynamic, effortless and joyful.

APPENDIX A

RESONANCE SCREENSHOTS

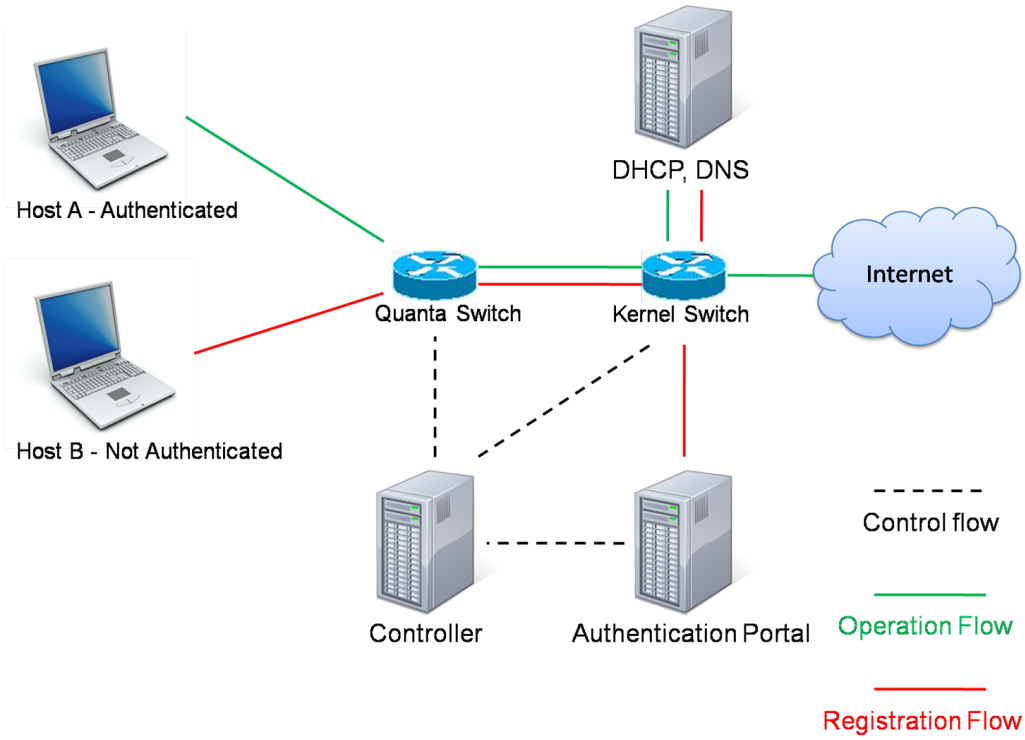

Figure 15: GEC7 Demo setup.

At the 7th GENI Engineering Conference (GEC7) [10], March 16, 2010, we demonstrated a working version of Resonance. The setup is shown in Figure 15. We had two switches connected to the controller. One of the switches was a Quanta switch in KACB, and the other one was a Kernel switch installed on a Linux box in CCB. Fiber path ensured direct connectivity between the two switches. As with our current deployment, control traffic was exchanged over a VLAN. To fully mimic an existing system, DHCP/DNS servers and web authentication portal are also part of the infrastructure. Having a live remote demo for our existing setup was tricky. We had to invoke a VM session inside VNC. As we see in the screenshots, there are two windows

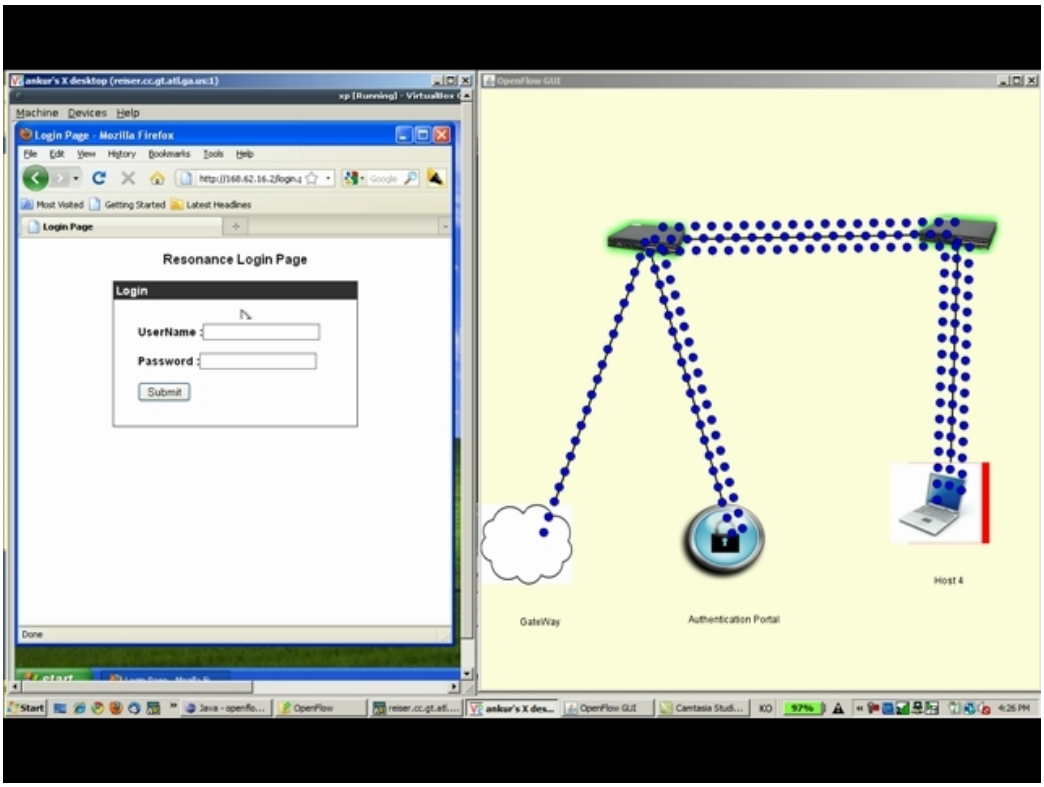

Figure 16: Resonance login page.

open. The window to left has a Windows XP VirtualBox guest VM running inside VNC. This VM's network interface is directly wired to a Resonance port and is totally isolated from the host network. The right window is a GUI front-end to NOX, called Envi [7]. We made custom modifications to the Envi code to enable flow visualization and other features.

The screenshots show three different stages of user authentication. Figure 16 is the main Resonance login page. A user gets redirected to this website on accessing any url (in this case, http://www.google.com). On right window, we see the corresponding traffic flow. There are two kinds of traffic flowing in the network. There is ARP traffic to the gateway (shown as a cloud). The other flow is the actual web traffic exchange between the host and the authentication portal. The red color on the host icon indicates that the host is in Registration state and is not authenticated. In the next step, the user successfully authenticates as depicted in

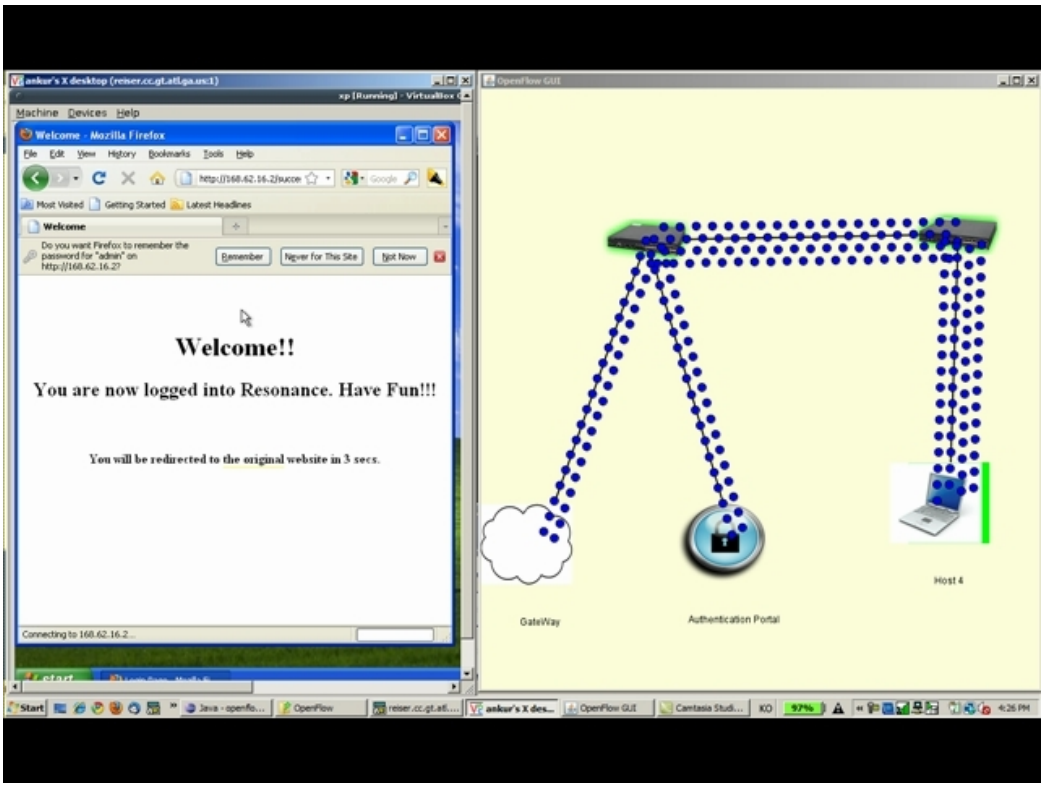

Figure 17: Successful login.

Figure 17. Note that the color of the host icon has now changed to green indicating that the login was successful. Finally, the user is redirected to the original Google website, after 3 seconds (Figure 18). Accordingly, the GUI window shows flow going only to the gateway. A video clip of Resonance showing all the steps is available at "http://www.youtube.com/watch?v=7gGeb1SEwgM".

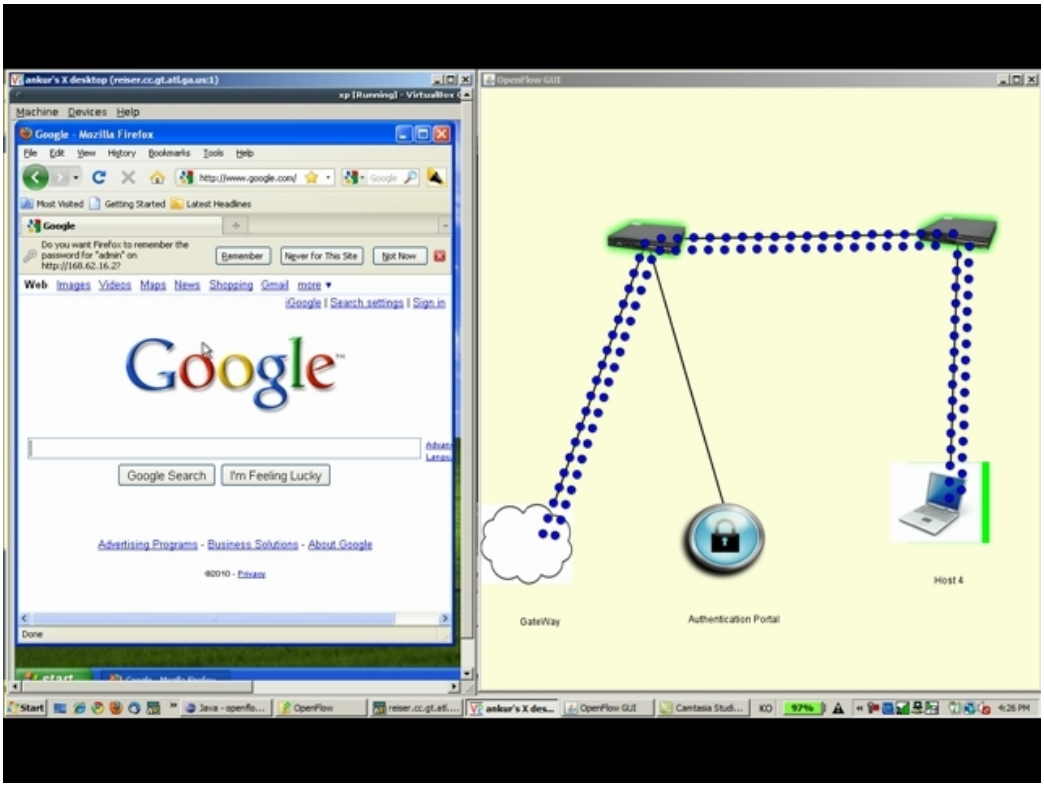

Figure 18: User redirected to original website.

REFERENCES

[1] "FlowVisor." http://www.openflowswitch.org/wk/index.php/FlowVisor, Aug. 2009.

[2] Bell, D. E. and Padula, L. J. L., "Secure computer systems: Unified exposition and multics interpretation." ESD-TR-75-306, Mitre Corporation, Bedford, MA, Mar 1976.

[3] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S., "Ethane : Taking control of the enterprise," in *SIGCOMM '07*, 2007.

[4] Casado, M., Koponen, T., Moon, D., and Shenker, S., "Rethinking packet forwarding hardware," in *Proc. Seventh ACM SIGCOMM HotNets Workshop*, Nov. 2008.

[5] "Cisco Application eXtension Platform Overview." http://www.cisco.com/en/US/prod/collateral/routers/ps9701/white_paper_c11_459082.html, Feb. 2009.

[6] "dnsmasq: DNS forwarder and DHCP server." http://www.thekelleys.org.uk/dnsmasq/doc.html, Feb. 2010.

[7] "An Extensible Network Visualization & Control Framework." http://www.openflowswitch.org/wp/gui/, Feb. 2010.

[8] Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., and van der Merwe, K., "The case for separating routing from routers," (Portland, OR), Sept. 2004.

[9] Feamster, N., Valancius, V., and Mundada, Y., "Bringing experimenters and external connectivity to GENI a.k.a. BGPMux, DTunnels." http://groups.geni.net/geni/wiki/BGPMux, Dec. 2009.

[10] "7th GENI Engineering Conference (GEC7)." http://www.geni.net/?p=1626, Mar. 2010.

[11] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H., "A clean slate 4D approach to network control and management," vol. 35, no. 5, pp. 41–54, 2005.

[12] Gu, G., Perdisci, R., Zhang, J., and Lee, W., "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *17th USENIX Security Symposium*, 2008.

[13] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S., "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 105–110, July 2008.

[14] Hinrichs, T., Gude, N., Casado, M., Mitchell, J., and Shenker, S., "Expressing and enforcing flow-based network security policies," tech. rep., Dec. 2008.

[15] "Human error blamed for network outages." `http://findarticles.com/p/articles/mi_m4153/is_6_59/ai_95572075`, Mar. 2010.

[16] "iptables: userspace program to configure Linux packet filtering ruleset." `http://www.netfilter.org/projects/iptables/index.html`, Nov. 2009.

[17] Knorr, K., "Dynamic access control through petri net workflows," in *Proc. 16th Annual Computer Security Applications Conference*, p. 159, 2000.

[18] Nayak, A., Reimers, A., Feamster, N., and Clark, R., "Resonance: Dynamic access control in enterprise networks," in *Proc. Workshop: Research on Enterprise Networking*, (Barcelona, Spain), Aug. 2009.

[19] Newman, P., Minshall, G., and Lyon., T. L., "IP Switching - ATM under IP," *IEEE/ACM Trans. Netw.*, vol. 6, no. 2, pp. 117–129, 1998.

[20] "OpenFlow Switch Consortium." `http://www.openflowswitch.org/`, Oct. 2008.

[21] Ramachandran, A., Feamster, N., and Vempala, S., "Filtering spam with behavioral blacklisting," in *Proc. 14th ACM Conference on Computer and Communications Security*, (Alexandria, VA), Oct. 2007.

[22] "SNAC: Simple Network Access Control." `http://www.openflowswitch.org/wp/snac/`, Mar. 2010.

[23] "Scanning Technology for Automated Registration, Repair and Response Tasks." `https://start.gatech.edu/`, Feb. 2009.

[24] Thomas, R. K. and Sandhu, R. S., "Towards a task-based paradigm for flexible and adaptable access control in distributed applications," in *Proc. 1992-1993 ACM SIGSAC New Security Paradigms Workshops, Little Compton, RI*, pp. 138–142, 1993.

[25] van der Merwe, J., Rooney, S., Leslie, I., and Crosby, S., "The Tempest - A Practical Framework for Network Programmability," *IEEE Network*, vol. 12, pp. 20–28, May 1998.

[26] "Configuring Dynamic Port VLAN Membership with VMPS." `http://www.cisco.com/univercd/cc/td/doc/product/lan/cat5000/rel_4_2/config/vmps.htm`, Mar. 2009.